Practice

- P1. Find the names of employees who are directly supervised by ‘Frank Wong’
- P2. Find the names of employees who work on at least one project
- P3. Find the names of employees who do not work on any project
- P4. Find the names of employees who have dependent named ‘Alice’
SQL Outline

- Data definition
- Query
  - Basic queries
  - More complex queries
  - Aggregate functions
- Data update
- View definition
AGGREGATE FUNCTIONS

- Include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**
- **Query 19**: Find the sum, maximum salary, the minimum salary, and the average salary among all employees.

Q19: SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY), AVG(SALARY) FROM EMPLOYEE
AGGREGATE FUNCTIONS (cont.)

Query 20: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

Q20: SELECT MAX(SALARY),
     MIN(SALARY),
     AVG(SALARY)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research'
AGGREGATE FUNCTIONS
(cont.)

- Queries 21 and 22: Retrieve the total number of employees in the company (Q21), and the number of employees in the 'Research' department (Q22).

  Q21: SELECT COUNT (*)
       FROM EMPLOYEE

  Q22: SELECT COUNT (*)
       FROM EMPLOYEE, DEPARTMENT
       WHERE DNO=DNUMBER AND
             DNAME='Research'

- Query 23: count the number of distinct salary values in the company?
GROUPING

In many cases, we want to apply the aggregate functions to subgroups of tuples in a relation
  - Group tuples to subgroups that have the same value for the grouping attribute(s)
  - The function is applied to each subgroup independently

SQL has a GROUP BY-clause for specifying the grouping attributes

Correspond to the grouping and aggregate function in relational algebra
Query 24: For each department, retrieve the department number, the number of employees in the department, and their average salary.

Q24: SELECT DNO, COUNT (*), AVG (SALARY) FROM EMPLOYEE GROUP BY DNO

- The SELECT-clause includes only the grouping attributes and the aggregate functions applied on each group of tuples

What happens if we include an attribute that’s not a group by attribute in the SELECT clause?

What happens if we do not include certain grouping attributes in the SELECT clause?
GROUPING (cont.)

- A selection or join condition can be used in conjunction with grouping
- **Query 25:** For each project, retrieve the project number, project name, and the number of employees who work on that project.

Q25: SELECT PNUMBER, PNAME, COUNT (*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME

The grouping and functions are applied *after* the selection/join condition is applied
THE HAVING-CLAUSE (cont.)

- The HAVING-clause is used for specifying a selection condition on groups (rather than on individual tuples).
- **Query 26:** For each project *on which more than two employees work*, retrieve the project number, project name, and the number of employees who work on that project.

```
Q26: SELECT PNUMBER, PNAME, COUNT(*)
     FROM PROJECT, WORKS_ON
     WHERE PNUMBER=PNO
     GROUP BY PNUMBER, PNAME
     HAVING COUNT(*) > 2
```

- Having clause can only be applied on grouping attribute and aggregate functions.
A Tricky Example

Q29: For each department with \( \geq 2 \) employees, find the department name, and the number of employees in that department that earns \( > 40000 \)

What does this query return?

```
SELECT dname, COUNT(ssn)
FROM department, employee
WHERE dnumber = dno
AND salary > 40000
GROUP BY dname
HAVING COUNT(ssn) \( \geq 2 \)
```
A Tricky Example

For each department with >= 2 employees, find the department name, and the number of employees in that department that earns > $40000

Find departments with >=2 employees
select dname, count(ssn)
from department, employee
where dnumber = dno
group by dname
having count(ssn)>2;

For departments with >=2 employees, find number of employees earning > 40000
SELECT  dname, COUNT(ssn)
FROM    employee, department
WHERE   dno = dnumber
AND     dno IN ( SELECT  dno
                  FROM    employee
                  GROUP BY dno
                  HAVING  COUNT(ssn) >= 2
              )
AND     salary > 40000
GROUP BY dname
ORDER BY

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s).
- **Query 15**: Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

Q15: SELECT DNAME, LNAME, FNAME, PNAME
FROM DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
WHERE DNUMBER=DNO
AND SSN=ESSN
AND PNO=PNUMBER
ORDER BY DNAME, LNAME

- The default order is in **ascending** order of values. We can specify the keyword **DESC** if we want a descending order; **ASC** explicitly for ascending order.
Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory.

SELECT  <attribute or function list>
FROM    <table list>
[WHERE  <selection or join condition>]
[GROUP BY <grouping attributes>]
[HAVING  <group condition>]
[ORDER BY <attribute list>]
Query Formulation Techniques - Division

- Example: Find lname of all employees who work on all projects controlled by department number 4
- Relational algebra: employees and their projects divided by projects controlled by department 4
- SQL: use NOT EXISTS and set difference to express division:
  projects controlled by department 4 – projects worked on by employee = empty
Query Formulation Techniques – Division (Cont.)

Q27:  SELECT fname, lname
       FROM employee e
       WHERE NOT EXISTS
             (SELECT pnumber
              FROM project
              WHERE dnum = 4
              MINUS
              SELECT pno
              FROM works_on
              WHERE essn = e.ssn);
Query Formulation Techniques – Most Number of

- Find the name of the departments with most number of employees
- The most number of employees per department can be found as:

```sql
SELECT MAX(COUNT(ssn))
FROM employee
GROUP BY dno
```

- Solution:

```sql
Q28: SELECT dname
FROM department, employee
WHERE dno = dnumber
GROUP BY dname
HAVING COUNT(ssn) = (SELECT MAX(COUNT(ssn))
FROM employee
GROUP BY dno)
```
Practice

- P5. Find the names of departments with 2 or more male employees
- P6. Find the names of employees who have more than 2 dependents and work on more than 2 projects (hint: intersection)
- P7. Find the names of employees with the most number of dependents
- P8. Find names of employees who work on all projects ‘John Smith’ is working on
Outline

- Data definition
- Working with SQL*Plus
- Query
- Data update
- View definition
Specifying Updates in SQL

- There are three SQL commands to modify the database: INSERT, DELETE, and UPDATE
**INSERT**

- It is used to add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command
- **Example:**

  ```sql
  INSERT INTO EMPLOYEE
  VALUES ('Richard','K','Marini', '653298653', '30-DEC-52', '98 Oak Forest,Katy,TX', 'M', 37000,'987654321', 4 )
  ```
Can specify explicitly the attribute names that correspond to the values in the new tuple

Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

```
INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
VALUES ('Richard', 'Marini', '653298653')
```

The constraints specified in the DDL commands are automatically enforced by the DBMS when updates are applied to the database
Another variation of INSERT allows insertion of *multiple tuples* resulting from a query into a relation.

**Example:** Suppose we want to create a temporary table that has the name, number of employees, and total salaries for each department. A table `DEPTS_INFO` is created by U3A, and is loaded with the summary information retrieved from the database by the query in U3B.

U3A: CREATE TABLE DEPTS_INFO
( DEPT_NAME VARCHAR(10),
  NO_OF_EMPS INTEGER,
  TOTAL_SAL INTEGER ) ;

U3B: INSERT INTO DEPTS_INFO ( DEPT_NAME,
  NO_OF_EMPS, TOTAL_SAL )
SELECT DNAME, COUNT(*) , SUM(SALARY) FROM DEPARTMENT, EMPLOYEE WHERE DNUMBER=DNO
GROUP BY DNAME ;

**Note:** The `DEPTS_INFO` table may not be up-to-date if we change the tuples in either the DEPARTMENT or the EMPLOYEE relations *after* issuing U3B. We have to create a view (see later) to keep such a table up to date.
DELETE

- Removes tuples from a relation
- Includes a WHERE-clause to select the tuples to be deleted
- Tuples are deleted from only one table at a time (unless CASCADE is specified on a referential integrity constraint)
- A missing WHERE-clause specifies that all tuples in the relation are to be deleted; the table then becomes an empty table
- The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause
- Referential integrity should be enforced
DELETE (cont.)

Examples:

U4A: DELETE FROM EMPLOYEE
    WHERE LNAME='Brown'

U4B: DELETE FROM EMPLOYEE
    WHERE SSN='123456789'

U4C: DELETE FROM EMPLOYEE
    WHERE DNO IN
    (SELECT DNUMBER
     FROM DEPARTMENT
     WHERE DNAME='Research')

U4D: DELETE FROM EMPLOYEE
UPDATE

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples *in the same relation*
- Referential integrity should be enforced
Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

U5: UPDATE PROJECT
   SET PLOCATION = 'Bellaire', DNUM = 5
   WHERE NUMBER=10
Example: Give all employees in the 'Research' department a 10% raise in salary.

U6: UPDATE EMPLOYEE
   SET SALARY = SALARY * 1.1
   WHERE DNO IN (SELECT DNUMBER
                  FROM DEPARTMENT
                  WHERE DNAME = 'Research')

In this request, the modified SALARY value depends on the original SALARY value in each tuple.

The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification.

The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification.
View Definition

- A view is a virtual relation that is derived from the relations in the conceptual schema or other existing views.
- A view does NOT exist in the physical form.
- A view can be used in queries as any "ordinary" relation. When it is used in a SELECT query, the virtual relation is computed first.

CREATE VIEW Emp_Activity
AS
( SELECT fname, lname, pname, hours
  FROM employee, works_on, project
  WHERE ssn = essn
  AND pno = pnumber
)

SELECT *
FROM Emp_Activity

DROP view Emp_Activity;